

DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG

```
DDDDDDDD  BBBB8888  GGGGGGGG  EEEEEEEEE  XX      XX  TTTTTTTTTT
DDDDDDDD  BBBB8888  GGGGGGGG  EEEEEEEEE  XX      XX  TTTTTTTTTT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BBB88888  GG      GG  EE      EE  XX      XX  TT
DD      DD  BBB88888  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG  GGGGGG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG  GGGGGG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DDDDDDDD  BBBB8888  GGGGGG  EEEEEEEEE  XX      XX  TT
DDDDDDDD  BBBB8888  GGGGGG  EEEEEEEEE  XX      XX  TT
                                         ....
                                         ....
                                         ....
                                         ....
```

```
RRRRRRRR  EEEEEEEEE  QQQQQQ
RRRRRRRR  EEEEEEEEE  QQQQQQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RRRRRRRR  EEEEEEEE  QQ      QQ
RRRRRRRR  EEEEEEEE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EE      EE  QQ      QQ
RR      RR  EEEEEEEEE  QQQQ  QQ
RR      RR  EEEEEEEEE  QQQQ  QQ
```

DBGEXT.REQ

Version: 'V04-000'

```
*****
*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
*  ALL RIGHTS RESERVED.
*
*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
*  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
*  TRANSFERRED.
*
*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
*  CORPORATION.
*
*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*****
```

WRITTEN BY

Rich Title

October 1983

MODIFIED BY

Robert Conti  
Edward FreedmanNovember 2, 1983  
December 12, 1983

MODULE FUNCTION

This module contains the definitions for the control blocks that are used in communications between DEBUG and the ADA multi-tasking run-time system. These same definitions will be extended for use in communication with the PPA multi-tasking system and other run-time systems, at a future time.



## EXTERNAL CONTROL BLOCK

An "External Control Block" is a data structure that can be used when DEBUG needs to call a routine that is not linked in as part of the DEBUG image.

For example, DEBUG will have commands to support ADA multi-tasking. However, DEBUG has no knowledge of the workings of the ADA multi-tasking system and the data structures that describe tasks. Instead, DEBUG will call a routine in the ADA multitasking system in the course of processing SHOW TASK, SET TASK, or any other command that requires knowledge about tasks.

There will be a single entry point, ADASDBGEXT, in the ADA multitasking system which is called by DEBUG. The External Control Block is the only parameter. Similarly, other multitasking run-time systems will have a single entry point, of the form <facility>SDBGEXT, with the entry point taking an External Control Block as its single parameter. In general, the External Control Block can be used as a means of communication with run-time systems that are not part of DEBUG. For example, in debugging the language SCAN we may want to allow the user to set breakpoints on events such as a SCAN pattern-match. The External Control Block will be the data structure that we use to communicate with the SCAN run-time system.

The DBGEXT\$V\_FACILITY\_ID field identifies which run-time system is being called. The VAX/VMS Facility code is used. Thus, it is assumed that there will be at most one DBGEXT entry point in the run-time code of any facility. Currently, legal values are the facility codes for ADA, PPA, and SCAN. This field may not actually be looked at (if desired, the run-time system may do a sanity check for the right value).

Since there are several functions we want each run-time system to perform for us, there is a DBGEXT\$W\_FUNCTION\_CODE field which specifies which function is to be performed.

All functions return a status code in the DBGEXT\$L\_STATUS field. For all functions, there is a DBGEXT\$L\_FLAGS field which can be used as a bitvector of flags. The exact use of these flags depends on the function.

The use of the remaining fields of the data structure depends upon the "FACILITY\_ID" field and upon the "FUNCTION\_CODE" field.

NOTE: DEBUG makes these calls with ASTs disabled. It is required that the run-time code not reenables ASTs during its execution.

The following illustrates the header of an External Control Block.  
The fields of an External Control Block are then illustrated  
for the case where the "FACILITY\_ID" is "ADA".

The following header is common to all External Control Blocks:

0	↑-----↑
	unused   V_FACILITY_ID   DBGEXT\$W_FUNCTION_CODE
	↑-----↑
1	
	DBGEXT\$L_STATUS
	↑-----↑
2	(some flags unused)   DBGEXT\$L_FLAGS
	↑-----↑
3	
	reserved for future use
	↑-----↑

The following illustrates the control block when the FACILITY\_ID field is "ADA". This control block is used for most functions (some functions, e.g. GET\_REGISTERS and SET\_REGISTERS use a longer control block, displayed later).

0	unused : V_FACILITY_ID : DBGEXT\$W_FUNCTION_CODE
1	DBGEXT\$L_STATUS
2	(some flags unused) DBGEXT\$L_FLAGS
3	reserved for future use
4	DBGEXT\$L_TASK_VALUE
5	DBGEXT\$L_TASK_NUMBER
6	unused : V_HOLD : V_STATE : DBGEXT\$W_SPECIFIED_FLAGS
7	DBGEXT\$V_PRIORITY
8	DBGEXT\$L_PRINT_ROUTINE
9	DBGEXT\$L_EVENT_ID



The following fields are present when the "FACILITY\_ID" field is "ADA" and the function code is

DBGEXT\$K\_GET\_REGISTERS,  
DBGEXT\$K\_SET\_REGISTERS,  
DBGEXT\$K\_SET\_ACTIVE.

For all other functions, the smaller block (without the register fields) is passed in.

0	unused : V_FACILITY_ID : DBGEXT\$W_FUNCTION_CODE
1	DBGEXT\$L_STATUS
2	(some flags unused) DBGEXT\$L_FLAGS
3	reserved for future use
4	DBGEXT\$L_TASK_VALUE
5	DBGEXT\$L_TASK_NUMBER
6	unused : V_HOLD : V_STATE : DBGEXT\$W_SPECIFIED_FLAGS
7	DBGEXT\$V_PRIORITY
8	DBGEXT\$L_PRINT_ROUTINE
9	DBGEXT\$L_EVENT_ID
10	DBGEXT\$L_R0
11	DBGEXT\$L_R1
12	DBGEXT\$L_R2
13	DBGEXT\$L_R3
14	DBGEXT\$L_R4
15	DBGEXT\$L_R5
16	DBGEXT\$L_R6
17	DBGEXT\$L_R7
18	DBGEXT\$L_R8
19	DBGEXT\$L_R9
20	DBGEXT\$L_R10
21	DBGEXT\$L_R11
22	DBGEXT\$L_AP

23	DBGEXT\$\$_FP
24	DBGEXT\$\$_SP
25	DBGEXT\$\$_PC
26	DBGEXT\$\$_PSL



## CONTROL BLOCK FIELDS

## FIELD DBGEXT\$HEADER\_FIELDS =

```

SET
DBGEXT$W_FUNCTION_CODE      = [ 0, 0, 16, 0],
DBGEXT$V_FACILITY_ID        = [ 0, 16, 12, 0],
! reserved                   = [ 0, 28, 4, 0],
DBGEXT$L_STATUS              = [ 1, 0, 32, 0],

DBGEXT$L_FLAGS               = [ 2, 0, 32, 0],
DBGEXT$V_ALL                 = [ 2, 0, 1, 0],%((WHAT WILL ALL DO?-tbs))%
DBGEXT$V_FULL                = [ 2, 1, 1, 0],%((explain FULL -tbs))%

DBGEXT$V_PSEUDO_GO           = [ 2, 2, 1, 0],
! Pseudo-go is set by the run-time system on return to DEBUG to
! indicate that DEBUG must do a pseudo-GO to accomplish the function.
! Used only for function SET_ACTIVE (see discussion under SET_ACTIVE).

DBGEXT$V_NO_HEADER           = [ 2, 3, 1, 0]
! Suppresses output of headers on a SHOW_TASK, SHOW_STATISTICS,
! or SHOW_DEADLOCKS.

! reserved                   = [ 0, 4, 28, 0],
! reserved                   = [ 4, 0, 32, 0],

```

TES;

## FIELD DBGEXT\$ADA\_FIELDS =

```

SET
DBGEXT$L_TASK_VALUE          = [ 4, 0, 32, 0],
DBGEXT$L_TASK_NUMBER         = [ 5, 0, 32, 0],
DBGEXT$W_SPECIFIED_FLAGS     = [ 6, 0, 16, 0],
DBGEXT$V_HOLD_SPECIFIED      = [ 6, 0, 1, 0],
DBGEXT$V_STATE_SPECIFIED     = [ 6, 1, 1, 0],
DBGEXT$V_PRIORITY_SPECIFIED  = [ 6, 2, 1, 0],
! reserved                   = [ 6, 3, 13, 0],
DBGEXT$V_STATE               = [ 6, 16, 4, 0],
DBGEXT$V_STATE_RUNNING       = [ 6, 16, 1, 0],
DBGEXT$V_STATE_READY         = [ 6, 17, 1, 0],
DBGEXT$V_STATE_SUSPENDED     = [ 6, 18, 1, 0],
DBGEXT$V_STATE_TERMINATED    = [ 6, 19, 1, 0],
DBGEXT$V_HOLD                = [ 6, 20, 1, 0],
! reserved                   = [ 6, 21, 11, 0],
DBGEXT$L_PRIORITY            = [ 7, 0, 32, 0],
DBGEXT$V_PRIORITY_00         = [ 7, 0, 1, 0],
DBGEXT$V_PRIORITY_01         = [ 7, 1, 1, 0],
DBGEXT$V_PRIORITY_02         = [ 7, 2, 1, 0],
DBGEXT$V_PRIORITY_03         = [ 7, 3, 1, 0],
DBGEXT$V_PRIORITY_04         = [ 7, 4, 1, 0],
DBGEXT$V_PRIORITY_05         = [ 7, 5, 1, 0],
DBGEXT$V_PRIORITY_06         = [ 7, 6, 1, 0],
DBGEXT$V_PRIORITY_07         = [ 7, 7, 1, 0],
DBGEXT$V_PRIORITY_08         = [ 7, 8, 1, 0],
DBGEXT$V_PRIORITY_09         = [ 7, 9, 1, 0],

```

```

DBGEXT$V_PRIORITY_10      = [ 7, 10, 1, 0],
DBGEXT$V_PRIORITY_11      = [ 7, 11, 1, 0],
DBGEXT$V_PRIORITY_12      = [ 7, 12, 1, 0],
DBGEXT$V_PRIORITY_13      = [ 7, 13, 1, 0],
DBGEXT$V_PRIORITY_14      = [ 7, 14, 1, 0],
DBGEXT$V_PRIORITY_15      = [ 7, 15, 1, 0],
DBGEXT$V_PRIORITY_16      = [ 7, 16, 1, 0],
DBGEXT$V_PRIORITY_17      = [ 7, 17, 1, 0],
DBGEXT$V_PRIORITY_18      = [ 7, 18, 1, 0],
DBGEXT$V_PRIORITY_19      = [ 7, 19, 1, 0],
DBGEXT$V_PRIORITY_20      = [ 7, 20, 1, 0],
DBGEXT$V_PRIORITY_21      = [ 7, 21, 1, 0],
DBGEXT$V_PRIORITY_22      = [ 7, 22, 1, 0],
DBGEXT$V_PRIORITY_23      = [ 7, 23, 1, 0],
DBGEXT$V_PRIORITY_24      = [ 7, 24, 1, 0],
DBGEXT$V_PRIORITY_25      = [ 7, 25, 1, 0],
DBGEXT$V_PRIORITY_26      = [ 7, 26, 1, 0],
DBGEXT$V_PRIORITY_27      = [ 7, 27, 1, 0],
DBGEXT$V_PRIORITY_28      = [ 7, 28, 1, 0],
DBGEXT$V_PRIORITY_29      = [ 7, 29, 1, 0],
DBGEXT$V_PRIORITY_30      = [ 7, 30, 1, 0],
DBGEXT$V_PRIORITY_31      = [ 7, 31, 1, 0],
DBGEXT$L_PRINT_ROUTINE    = [ 8, 0, 32, 0],
DBGEXT$L_EVENT_ID        = [ 9, 0, 32, 0],
TES;

```

## FIELD DBGEXT\$REG\_FIELDS =

```

SET
DBGEXT$L_R0      = [10, 0, 32, 0],
DBGEXT$L_R1      = [11, 0, 32, 0],
DBGEXT$L_R2      = [12, 0, 32, 0],
DBGEXT$L_R3      = [13, 0, 32, 0],
DBGEXT$L_R4      = [14, 0, 32, 0],
DBGEXT$L_R5      = [15, 0, 32, 0],
DBGEXT$L_R6      = [16, 0, 32, 0],
DBGEXT$L_R7      = [17, 0, 32, 0],
DBGEXT$L_R8      = [18, 0, 32, 0],
DBGEXT$L_R9      = [19, 0, 32, 0],
DBGEXT$L_R10     = [20, 0, 32, 0],
DBGEXT$L_R11     = [21, 0, 32, 0],
DBGEXT$L_AP      = [22, 0, 32, 0],
DBGEXT$L_FP      = [23, 0, 32, 0],
DBGEXT$L_SP      = [24, 0, 32, 0],
DBGEXT$L_PC      = [25, 0, 32, 0],
DBGEXT$L_PSL     = [26, 0, 32, 0],
TES;

```

## LITERAL

```

DBGEXT$K_HEADER_SIZE      = 4,      ! Size of header in longwords
DBGEXT$K_ADA_SIZE1        = 10,     ! Size of block for ADA (without regs)
DBGEXT$K_ADA_SIZE2        = 27,     ! Size of block for ADA (with regs)
DBGEXT$K_MAX_SIZE         = 27,     ! Max of above sizes

```

## MACRO

```

DBGEXT$CONTROL_BLOCK = BLOCK [DBGEXT$K_MAX_SIZE]
FIELD ( DBGEXT$HEADER_FIELDS,

```

DBGEXT.REQ;1

16-SEP-1984 16:48:48.58 Page 9

DBGEXT\$ADA\_FIELDS.  
DBGEXT\$REG\_FIELDS ) %;

Generally, multiple priorities and states are valid as input when calling the ADA run time system but are not valid as output values on return from the call. Therefore, the following constants are provided for convenience in setting and testing the contents of the fields DBGEXTSV\_STATE and DBGEXTSV\_PRIORITY. They define the only possible values of the respective fields when multiple priorities and states are not allowed. Constants for DBGEXTSV\_HOLD are provided for completeness.

## LITERAL

DBGEXTSK_MIN_STATE	= ,	%((superfluous? -tbs))%
DBGEXTSK_MAX_STATE	= ,	
DBGEXTSS_STATE	= 4,	! size of DBGEXTSV_STATE
DBGEXTSK_STATE_RUNNING	= 1 ^ 0,	! values for DBGEXTSV_STATE
DBGEXTSK_STATE_READY	= 1 ^ 1,	
DBGEXTSK_STATE_SUSPENDED	= 1 ^ 2,	
DBGEXTSK_STATE_TERMINATED	= 1 ^ 3,	
DBGEXTSS_HOLD	= 1,	! size of DBGEXTSV_HOLD
DBGEXTSK_HOLD	= 1 ^ 0,	! values for DBGEXTSV_HOLD
DBGEXTSS_PRIORITY	= 32,	! size of DBGEXTSV_PRIORITY
DBGEXTSK_PRIORITY_00	= 1 ^ 0,	! values for DBGEXTSV_PRIORITY
DBGEXTSK_PRIORITY_01	= 1 ^ 1,	
DBGEXTSK_PRIORITY_02	= 1 ^ 2,	
DBGEXTSK_PRIORITY_03	= 1 ^ 3,	
DBGEXTSK_PRIORITY_04	= 1 ^ 4,	
DBGEXTSK_PRIORITY_05	= 1 ^ 5,	
DBGEXTSK_PRIORITY_06	= 1 ^ 6,	
DBGEXTSK_PRIORITY_07	= 1 ^ 7,	
DBGEXTSK_PRIORITY_08	= 1 ^ 8,	
DBGEXTSK_PRIORITY_09	= 1 ^ 9,	
DBGEXTSK_PRIORITY_10	= 1 ^ 10,	
DBGEXTSK_PRIORITY_11	= 1 ^ 11,	
DBGEXTSK_PRIORITY_12	= 1 ^ 12,	
DBGEXTSK_PRIORITY_13	= 1 ^ 13,	
DBGEXTSK_PRIORITY_14	= 1 ^ 14,	
DBGEXTSK_PRIORITY_15	= 1 ^ 15,	
DBGEXTSK_PRIORITY_16	= 1 ^ 16,	
DBGEXTSK_PRIORITY_17	= 1 ^ 17,	
DBGEXTSK_PRIORITY_18	= 1 ^ 18,	
DBGEXTSK_PRIORITY_19	= 1 ^ 19,	
DBGEXTSK_PRIORITY_20	= 1 ^ 20,	
DBGEXTSK_PRIORITY_21	= 1 ^ 21,	
DBGEXTSK_PRIORITY_22	= 1 ^ 22,	
DBGEXTSK_PRIORITY_23	= 1 ^ 23,	
DBGEXTSK_PRIORITY_24	= 1 ^ 24,	
DBGEXTSK_PRIORITY_25	= 1 ^ 25,	
DBGEXTSK_PRIORITY_26	= 1 ^ 26,	
DBGEXTSK_PRIORITY_27	= 1 ^ 27,	
DBGEXTSK_PRIORITY_28	= 1 ^ 28,	
DBGEXTSK_PRIORITY_29	= 1 ^ 29,	
DBGEXTSK_PRIORITY_30	= 1 ^ 30,	
DBGEXTSK_PRIORITY_31	= 1 ^ 31,	



DBGEXT.REQ;1

16-SEP-1984 16:48:48.58<sup>F 14</sup> Page 11

## FACILITY CODES

The following are the possible values of the DBGEXTSV.FACILITY\_ID field.  
These correspond to the different run-time system we are  
communicating with.

ADAS\_FACILITY  
PPAS\_FACILITY  
SCNS\_FACILITY

\*QUES\* %((-tbs))%

Do PPA and SCAN have facility mnemonics and codes? Are the  
above guesses correct?

## FUNCTION CODES

The following are the possible values of the DBGEXTSW FUNCTION CODE field when the contents of the FACILITY\_ID field is ADASFACILITY. These correspond to the functions that the ADA run-time system will be asked to perform.

## Summary of the defined Function codes

DBGEXTSK\_MIN\_FUNCT = 1, ! For CASE bounds

These are used to obtain and convert task values

DBGEXTSK\_CVT\_VALUE\_NUM = 1,  
DBGEXTSK\_CVT\_NUM\_VALUE = 2,  
DBGEXTSK\_NEXT\_TASK = 3,

These are used to ask ADA to display task information

DBGEXTSK\_SHOW\_TASK = 4,  
DBGEXTSK\_SHOW\_STATISTICS = 5,  
DBGEXTSK\_SHOW\_DEADLOCK = 6,

These are used to get and set various attributes of one or more tasks

## Task state

DBGEXTSK\_GET\_STATE = 7,  
DBGEXTSK\_GET\_ACTIVE = 8,  
DBGEXTSK\_SET\_ACTIVE = 9,  
DBGEXTSK\_SET\_TERMINATE = 10,  
DBGEXTSK\_SET\_HOLD = 11,

## Task priority

DBGEXTSK\_GET\_PRIORITY = 12,  
DBGEXTSK\_SET\_PRIORITY = 13,  
DBGEXTSK\_RESTORE\_PRIORITY = 14,

## Task registers

DBGEXTSK\_GET\_REGISTERS = 15,  
DBGEXTSK\_SET\_REGISTERS = 16,

These are used to control definable events

DBGEXTSK\_ENABLE\_EVENT = 17,  
DBGEXTSK\_DISABLE\_EVENT = 18,

DBGEXTSK\_MAX\_FUNCT = 18; ! For CASE bounds

## LITERAL

A minimum task code is defined for CASE statement bounds.

DBGEXTSK\_MIN\_FUNCT = 1.

CVT\_VALUE\_NUM takes a task value and converts it to a task number.

INPUT - The task value is placed in the DBGEXTSL\_TASK\_VALUE field.

OUTPUT - The task number is returned in the DBGEXTSL\_TASK\_NUMBER field.

(If the task does not exist, this function returns  
status STSSK\_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%  
%(VALUE IS NOT LEGAL OR ACCVIO? -tbs))%

DBGEXTSK\_CVT\_VALUE\_NUM = 1.

CVT\_NUM\_VALUE takes a task number and converts it to a task value.

INPUT - The task number is placed in the DBGEXTSL\_TASK\_NUMBER field.

OUTPUT - The task value is returned in the DBGEXTSL\_TASK\_VALUE field.

(If the task does not exist, this function returns  
status STSSK\_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%

DBGEXTSK\_CVT\_NUM\_VALUE = 2.

NEXT\_TASK gives a task value and asks ADA to specify the 'next' task. The ordering of tasks is up to the ADA run-time system. The only requirement on order is that if we start with any task, and repeatedly ask for the 'next' without giving the user program control in between, then we will cycle through all the tasks and return to the task we started with. If selection criteria are imposed, then we will cycle through all tasks which match that criteria.

INPUTS - The task value is placed in the DBGEXTSL\_TASK\_VALUE field.

If the TASK\_VALUE field is zero (implying the NULL task) the next task will be the main task of the program.

The ALL flag is ignored, ADA will consider it on by default.

The set of tasks to cycle through can be restricted by imposing a selection criteria. The PRIORITY, and/or STATE, and/or HOLD fields can contain values which a task must match to be part of the set (e.g. SHOW TASK/PRI=3/HOLD/STATE=READY). When such a restriction is desired, the DBGEXTSV\_xxx SPECIFIED bits must be set accordingly. If no restriction is desired, the SPECIFIED bits must be zero. A task must match all the criteria which are specified to be part of the set.



!%((Multiple PRI and STATE can be given as these are bit fields -tbs))%

OUTPUT - The "next" task value is returned in DBGEXT\$TASK\_VALUE.

DBGEXT\$K\_NEXT\_TASK = 3.

SHOW\_TASK is used to request that ADA display information about a specified task.

INPUTS - The task value is placed in the DBGEXT\$TASK\_VALUE field.

The address of a print routine that ADA is to call, to display the information, is placed in the field DBGEXT\$PRINT\_ROUTINE (see DBG\$PRINT\_ROUTINE below).

If the DBGEXT\$V\_FULL bit is set, more detailed information is displayed.

OUTPUT - none.

DBGEXT\$K\_SHOW\_TASK = 4.

SHOW\_STATISTICS requests that the ADA run-time system display statistics about the overall state of the multitasking system.

INPUTS - The address of a print routine is given in the field DBGEXT\$PRINT\_ROUTINE.

If the DBGEXT\$V\_FULL bit is set, more detailed information is displayed.

OUTPUT - none.

DBGEXT\$K\_SHOW\_STAT = 5.

SHOW\_DEADLOCK requests that the ADA run-time system display information about deadlocks within the multitasking system.

INPUTS - The address of a print routine is given in the field DBGEXT\$PRINT\_ROUTINE.

If the DBGEXT\$V\_FULL bit is set, more detailed information is displayed.

OUTPUT - none.

DBGEXT\$K\_SHOW\_DEADLOCK = 6.

GET\_STATE inquires about the "state" and HOLD condition of a task. The "state" can be one of RUNNING, READY, SUSPENDED, TERMINATED. The state codes are defined below.

INPUT - The task value is placed in the DBGEXT\$TASK\_VALUE field.

OUTPUTS - A code representing the state is returned in the  $\%((V\_STATE - tbs))\%$  DBGEXT\$STATE field.

The DBGEXT\$V\_HOLD field is also set if the task is on HOLD.

DBGEXT\$K\_GET\_STATE = 7,

GET\_ACTIVE obtains the task value of the active task. (The active task is that task in whose context (stack and register set) DEBUG is executing. This is contrasted with the "visible task" -- the task whose register set is temporarily in use by DEBUG as a default for the purposes of SHOW CALLS, EXAMINE, etc.).

INPUTS - none

OUTPUT - The task value of the active task is returned in DBGEXT\$TASK\_VALUE.

$\%((\text{Can the active task be the null task? -tbs}))\%$

DBGEXT\$K\_GET\_ACTIVE = 8,

SET\_ACTIVE requests the run-time system to switch the active task to that given in DBGEXT\$TASK\_VALUE. The "long form" DBG control block is used. The registers provided by DEBUG in the control block are those of the (currently) active task. The run-time system uses these to save the registers of the active task. It may also modify this register set, (currently only the PC and PSL). When this call returns, DEBUG should use the possibly-modified register values as the active register set. If the PSEUDO GO bit is set, DEBUG should then perform the actions of a normal GO, except that ASTs are left disabled. This "pseudo-GO" will enter special run-time code that will switch-out the currently active task, switch-in the requested active task, and reinvoked DEBUG in that task. (A special event code is assigned to this "reinvoke DEBUG event". The reinvocation event signifies to DEBUG that certain components of its state are to be gotten from values saved from DEBUG's prior incarnation, not those at the reinvocation event. One such saved state component is the "AST enablement" status - whether ASTs were enabled when DEBUG was invoked.) Despite these gyrations, to the user typing DBG> SET TASK/ACTIVE T1, it appears he has entered a simple command immediately followed by a DBG> prompt.

INPUTS - The task value of the to-become-active task is set in DBGEXT\$TASK\_VALUE.

The registers of the (currently) active task are stored in fields DBGEXT\$R0 through DBGEXT\$PSL.

OUTPUTS - The register set of the new active task, as

modified by the run-time system, in DBGEXT\$\$\_RO  
through DBGEXT\$\$\_PSL.

The DBGEXT\$\$\_PSEUDO\_GO flag may be set, in which case,  
DEBUG should perform a "pseudo go" operation.

DBGEXT\$\$\_SET\_ACTIVE = 9,

SET\_TERMINATE is used to cause ADA to terminate a task. It is used  
to implement the command SET TASK/TERMINATE.

INPUTS - The task value is placed in the DBGEXT\$\$\_TASK\_VALUE field.

If the TASK\_VALUE field is zero and the ALL flag  
is set, then the function is done for all tasks.

OUTPUT - none

DBGEXT\$\$\_SET\_TERMINATE = 10,

SET\_HOLD is used to put a task on hold or to release a task that was  
previously put on hold. It is used to implement the command  
SET TASK/HOLD which leaves the state of a task as-is, except that each  
task is marked HOLD.

INPUTS - The task value is placed in the DBGEXT\$\$\_TASK\_VALUE field.

If the TASK\_VALUE field is zero and the ALL flag  
is set, then the function is done for all tasks.

((( Will the /ALL selection criteria be used for the SET\_xxx codes? -tbs)))

The desired status of HOLD is placed into the  
DBGEXT\$\$\_HOLD field. (1 => HOLD, 0 => RELEASE)

(((Is the request 1=>1 or 0=>0 legal? -tbs)))

OUTPUT - none

DBGEXT\$\$\_SET\_HOLD = 11,

GET\_PRIORITY inquires about the priority of a specified task.

INPUT - The task value is placed in the DBGEXT\$\$\_TASK\_VALUE field.

OUTPUT - The priority is returned in the DBGEXT\$\$\_PRIORITY field.

DBGEXT\$\$\_GET\_PRIORITY = 12,

SET\_PRIORITY is used to set the priority of a specified task.

INPUTS - The task value is placed in the DBGEXT\$\$\_TASK\_VALUE field.

If the TASK\_VALUE field is zero and the ALL flag is set, then the function is done for all tasks.

The desired priority is placed in the DBGEXT\$W\_PRIORITY field.

OUTPUT - none.

DBGEXT\$K\_SET\_PRIORITY = 13,

RESTORE\_PRIORITY is used to restore the priority of a task back to its normal value (as it would be without DEBUG intervention).

INPUTS - The task value is placed in the DBGEXT\$L\_TASK\_VALUE field.

If the TASK\_VALUE field is zero and the ALL flag is set, then the function is done for all tasks.

OUTPUT - none.

DBGEXT\$K\_RESTORE\_PRIORITY = 14,

GET\_REGISTERS is used to obtain the register set of a task.

INPUT - The task value is placed in the DBGEXT\$L\_TASK\_VALUE field.

OUTPUTS - The register values are returned in the DBGEXT\$L\_R0 through DBGEXT\$L\_PSL fields.

NOTE: Only DEBUG knows the register set of the active task hence, this call is invalid for the active task. A return status of ST\$K\_SEVERE is returned.

DBGEXT\$K\_GET\_REGISTERS = 15,

SET\_REGISTERS is used to change the register values of a task. This may be needed, for example, in SET TASK T;DEPOSIT R5 = 0;GO

INPUTS - The task value is placed in the DBGEXT\$L\_TASK\_VALUE field.

The register values are placed in the DBGEXT\$L\_R0 through DBGEXT\$L\_PSL fields.

OUTPUT - none.

NOTE: Only DEBUG knows the register set of the active task hence, this call is invalid for the active task. A return status of ST\$K\_SEVERE is returned.

DBGEXT\$K\_SET\_REGISTERS = 16,

! ENABLE\_EVENT is used during processing of a "SET BREAK/EVENT=" or



! "SET TRACE/EVENT=" command to enable reporting of a given kind of event.

INPUTS - The DBGEXT\$L\_EVENT\_ID field contains a code identifying the event being enabled. The possible values of this code are defined below.

The DBGEXT\$L\_TASK\_VALUE field contains a task value further qualifying the event being enabled. This may be zero if the "ALL" flag is lit.

For example,  
if we are enabling "task termination" and we supply a task value, then we only want to break on termination of that task. If we enable "task termination" events and set the ALL flag, we want to be notified of any task termination.

OUTPUT - none

DBGEXT\$K\_ENABLE\_EVENT = 17,

! DISABLE\_EVENT is used during processing of a "CANCEL BREAK/EVENT=" or "CANCEL TRACE/EVENT=" command to disable reporting of a given kind of event.

INPUTS - The DBGEXT\$L\_EVENT\_ID field contains a code identifying the event being disabled. The possible values of this code are defined below.

The DBGEXT\$L\_TASK\_VALUE field contains a task value further qualifying the event being disabled. This may be zero if the "ALL" flag is lit.

OUTPUT - none

DBGEXT\$K\_DISABLE\_EVENT = 18,

! A maximum task code is defined for CASE statement bounds.

DBGEXT\$K\_MAX\_FUNCT = 18;

## COMPLETION STATUS

The run time system has two means of providing a completion status -- the return value of the function and the contents of DBGEXT\$L\_STATUS.

Function Return Value --

The run time system should, as its first action, attempt to read and verify the field DBGEXT\$V FACILITY\_ID in DBGEXT\$CONTROL\_BLOCK. Optionally, it may also PROBE the control block for read/writability. If the FACILITY\_ID is correct, the run time system should eventually return:

ST\$K\_SUCCESS                      - service successfully completed

Otherwise, the run time system should immediately return:

ST\$K\_SEVERE                      - service failed

This helps to insure that an incorrect External Control Block will be detected before it is written to.

Contents of DBGEXT\$L\_STATUS --

All other status and error conditions will be placed in the STATUS field of the control block. The possible values of the STATUS field are a composite of severity level and message number. Only two severity values are used. They are given by ST\$V\_SEVERITY:

ST\$K\_SUCCESS                      - service successfully completed

In this case the message number (ST\$V\_MSG\_NO) is zero.

ST\$K\_ERROR                      - service failed

In this case the message number (ST\$V\_MSG\_NO) is one of the following:

## LITERAL

DBGEXT\$K\_FUNCTION\_NOT\_IMP    = 0,

! The function requested is not implemented by the facility.

DBGEXT\$K\_TASK\_NOT\_EXIST      = 1,

! Task number cannot be translated to a task value because the task does not exist. Or task value does not point to a currently existing task ! (this cannot always be detected).

DBGEXT\$K\_TASK\_IS\_ACTIVE      = 2,

! Returned on a SET\_REGISTER or GET\_REGISTER function for the active task. ! The run time system cannot access the registers of the active task.

DBGEXT\$K\_TASK\_IS\_NULL        = 3;

! Returned on a SET\_ACTIVE function for the null task.

## PRINT ROUTINE INTERFACE

The following defines how to use the DEBUG print routine whose address is given in the DBGEXT\$\$\_PRINT\_ROUTINE field.

```
BIND
  DBG$PRINT_ROUTINE = .control_block [ DBG$$_PRINT_ROUTINE ];
```

```
DBG$PRINT_ROUTINE ( NEW_LINE,
                   STRING_TO_PRINT,
                   FAO_ARG_1,
                   FAO_ARG_2,...
                   FAO_ARG_n) : NOVALUE
```

NEW\_LINE - this can have one of two values:

- 0 - Place the given string in the output buffer.
- 1 - If the given string is non-zero, first place it in the buffer. In all cases, output the buffer to the screen.

## STRING\_TO\_PRINT

- this is a pointer to a counted ascii string  
E.g., UPLIT (%ASCIC 'Output this text')  
This may be zero if the ACTION\_CODE is 'NEWLINE'.

There may be FAO arguments following the string.  
The string thus may contain embedded FAO commands  
such as '!AC', '!SL', and so on.

%((FIXUP - THIS EXTENSION IS NOT GOOD!! -tbs))%

In addition, there will be a DEBUG-specific extension  
to FAO which can be used for symbolizing addresses.  
There will be a new command '!SA' for "symbolize address".  
This indicates that the corresponding FAO argument  
is an address. It's symbolization is to be embedded into  
the string.

FAO\_ARG1 through FAO\_ARGn - optional parameters for FAO arguments.

Example: suppose FOO\L is located at address 200. Then:

```
DBG$PRINT_ROUTINE (DBGEXT$$_NEWLINE,
                  UPLIT (%ASCIC 'Task switch at location !SA'),
                  200);
```

This would output:

"Task switch at location FOO\L"



## EVENT ID

The following define the possible values of the DBGEXTSL\_EVENT\_ID field.  
These are the predefined events that we can break or trace on.

## LITERAL

```
DBGEXTSK_MIN_EVENT_CODE      = 0,
DBGEXTSK_INVOKE_DEBUG        = 0,      ! Unconditional DEBUG invocation

DBGEXTSK_TASK_ACTIVATION     = 1,      ! First transition of a task to RUNNING
DBGEXTSK_TASK_SUSPENSION     = 2,      ! Transition from RUNNING to SUSPENDED
DBGEXTSK_TASK_SWITCH_FROM    = 3,      ! Transition from RUNNING to some state
DBGEXTSK_TASK_SWITCH_TO      = 4,      ! Transition from some state to RUNNING
DBGEXTSK_TASK_TERMINATION    = 5,      ! Any kind of termination

! Ada specific tasking codes:
DBGEXTSK_TASK_ABORT_TERM     = 6,      ! Termination by abort
DBGEXTSK_TASK_EXCEP_TERM     = 7,      ! Termination by unhandled exception
DBGEXTSK_TASK_EXCEP_REND     = 8,      ! Exception propagating out of rendezvous
DBGEXTSK_TASK_ENTRY_CALL     = 9,      ! Executing an entry call
DBGEXTSK_TASK_ACCEPT         = 10,     ! Executing an accept
DBGEXTSK_TASK_SELECT         = 11,     ! Executing a select

DBGEXTSK_MAX_EVENT_CODE      = 11;
```

## EVENT CONTROL BLOCK

The Event Control Block is the data structure that the ADA (or other) facility passes to DEBUG when it signals that a given event has occurred.

For example, if you do a SET BREAK/ADAEVENT=TASK\_SWITCH\_TO, then when a task switch occurs, the ADA run-time system will signal the special signal DBGS\_EVENT. A pointer to an "Event Control Block" is passed as the "FAO argument" of DBGS\_EVENT. (E.g., LIBSSIGNAL (DBGS\_EVENT, 1, .EVENT CONTROL BLOCK). (Note that this condition cannot properly be an SSS condition because they are not allowed to have FAO arguments other than PC and PSL (except for the hardware conditions). Hence, the facility DBG was chosen. This condition is a DEBUG-defined condition that anyone can signal. The FAO count of 1 is required so that the message conforms to a legal format for a message vector.) Through proper use of the SEVERITY field and the NOMESSAGE bit in the condition, the signaller can be assured that events will be "reflected" by Traceback should DEBUG not be mapped into the image (for some reason). So there really are no restrictions on when this condition can be signalled.

The control block contains a code indicating the facility that has originated the event and another code to indicate what event has occurred. It also contains message text to be output announcing the event.

The following illustrates the Event Control Block:

0	DBGEXT\$\$_EVNT_FACILITY_ID
1	DBGEXT\$\$_EVNT_EVENT_ID
2	DBGEXT\$\$_EVNT_FLAGS
3	DBGEXT\$\$_EVNT_MESSAGE_TXT
4	DBGEXT\$\$_EVNT_ARG_COUNT
5	DBGEXT\$\$_EVNT_ARG_VECT
	...

FIELD DBGEXT\$\$\_EVNT\_FIELDS =

```

SET
DBGEXT$$_EVNT_FACILITY_ID = [0, 0, 32, 0],
DBGEXT$$_EVNT_EVENT_ID   = [1, 0, 32, 0],
DBGEXT$$_EVNT_FLAGS      = [2, 0, 32, 0],
DBGEXT$$_EVNT_MORE_TEXT  = [2, 0, 1, 0],           ! Flag bit 0
DBGEXT$$_EVNT_REENTRY    = [2, 1, 1, 0],           ! Flag bit 1

DBGEXT$$_EVNT_MESSAGE_TXT = [3, 0, 32, 0],
DBGEXT$$_EVNT_ARG_COUNT   = [4, 0, 32, 0],
DBGEXT$$_EVNT_ARG_VECT    = [5, 0, 0, 0]
TES;
```

## LITERAL

DBGEXT\$K\_EVNT\_BASE\_SIZE = 5;

## MACRO

DBGEXT\$EVENT CONTROL BLOCK(NUM\_ARGS) =  
BLOCK [DBGEXT\$K\_BASE\_SIZE + NUM\_ARGS ,LONG]  
FIELD (DBGEXT\$EVNT\_FIELDS)%;

## Explanation of fields:

- FACILITY\_ID field:** The code for the facility signaling the event. If the CUST\_DEF bit is set the event is a "user event". Otherwise, the only supported codes are ADA, PPA, and scan.
- EVENT\_ID field:** This field contains the event code. Event codes are numbered from 1 within each facility. Event code 0 is reserved in all facilities. It represents the unconditional event, that is, unconditional DEBUG entry. If the EVENT\_ID field is zero, the REENTRY bit is checked.
- MESSAGE\_TXT field:** This is a pointer to a counted ascii string. The string represents a message to be printed when the event occurs and is formatted as an "fao control string". The string may take FAO arguments. The string may also contain the DEBUG extension to FAO, '!SA', in order to symbolize an address. This extension is described above. NOTE: if this field is 0, it indicates that there is no message.
- ARG\_COUNT field:** Count of the number of FAO arguments that go with the text.
- ARG\_VECT field:** A vector of FAO arguments.
- MORE\_TEXT flag:** If this flag is TRUE, it indicates that DEBUG is to return control at the point of the signal after displaying the message. This is to be used for output of multi-line messages. (I.e., the run-time system should then resignal the event with the next line of message text in the MESSAGE\_TXT field).
- REENTRY flag:** If this flag is TRUE, then this event is a DEBUG-reentry event that has occurred after a PSEUDO GO. DEBUG is thereby instructed to restore certain components of its state from the values they had at DEBUG's last incarnation (e.g. AST enablement).

DBGEXT.REQ;1

16-SEP-1984 16:48:48.58<sup>H 15</sup> Page 26

For this flag to be checked by DEBUG, the  
EVENT\_ID field MUST BE ZERO, thus indicating  
unconditional entry to DEBUG.



## REGISTERING EVENTS WITH DEBUG

DEBUG's event handling feature is available to user programs as well as Digital software. DEBUG maintains an event table for each facility that chooses to register its events with DEBUG.

Registering an event with DEBUG is very simple. The facility need only signal the following signal after DEBUG has been invoked in an image:

```
LIB$SIGNAL(DBG$_REGISTER_EVENTS,  
            first_event_condition,  
            second_event_condition,  
            etc.
```

A list of event conditions is chained below a master condition of DBG\$\_REGISTER\_EVENTS. This signal may be raised as many times as desired to add more events to DEBUG's event table. Since DEBUG derives the facility number from the event condition, events for different facilities may be registered with the same signal.

The event conditions appearing in the message vector must be defined in the facilities message file. The string defined in the message file is the string that DEBUG will use to name the event.

For example, suppose we wish to add an event of PLI\$\_TASK\_SWITCH. The following would do it:

1. Add to PLI's message file:  
    PLI\$\_FACILITY = xxx  
    TASK\_SWITCH "TASK\_SWITCH"
2. Register the event with DEBUG  
    LIB\$SIGNAL(DBG\$\_REGISTER\_EVENTS, PLI\$\_TASK\_SWITCH)

After the registration, any user can then type  
SET BREAK/EVENT=PLI\$\_TASK\_SWITCH

A command SET EVENT/FACILITY='PLI\$ ' can be used so the facility prefix can be omitted, e.g. SET BREAK/EVENT=TASK\_SWITCH. This will then not be confused with an Ada task switch. SET EVENT/NOFACILITY will eliminate the automatic prefixing of event names.

To simplify the registration of events by facilities, any facility should provide an entry point that users can call from the DEBUGGER to load the events of that facility. To load PLI's events, then, a user would merely type

```
DBG> CALL PLI$_LOAD_EVENTS
```

\*\* Obviously, Ada's events should be registered with this same general mechanism



0075 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY